云资源自动化 for Terraform

结合 Devops

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

结合 Devops
代码管理
持续集成与部署
Terraform 在 Github 中的应用

版权所有：腾讯云计算（北京）有限责任公司

结合 Devops
代码管理

最近更新时间：2023-09-21 20:57:32

基本概念

GitOps

GitOps 是 Weaveworks 提出的一种持续交付方式，它的核心思想是将应用系统的声明性基础架构和应用程序存放在 Git 版本库中。将 Git 作为交付流水线的核心，每个开发人员都可以提交拉取请求（Pull Request）并使用 Git 来加速和简化 Terraform 的应用程序部署和运维任务。通过使用像 Git 这样的简单工具，开发人员可以更高效地将注意力集中在创建新功能而不是运维相关任务上（例如，应用系统安装、配置、迁移等）。

Terraform 根模块

根配置（根模块）是您在其中运行 Terraform CLI 的工作目录。

根模块标准：

- **最大限度地减少每个根模块中的资源数量**

  请防止单个根配置过大，同时还要防止同一目录和状态下存储的资源过多。每次运行 Terraform 时，特定根配置中的所有资源都会刷新。如果单个状态中包含的资源过多，则可能会导致执行缓慢。一般规则：在单个状态下，不要添加超过 100 个资源（最好不要超过十几个）。
- **为每个应用使用单独的目录**

  如需相互独立地管理应用和项目，请将每个应用和项目的资源放在其各自的 Terraform 目录中。服务可能表示特定应用或通用服务，例如共享网络。将特定服务的所有 Terraform 代码嵌套在一个目录（包括子目录）下。

项目结构

将服务的 Terraform 配置拆分为两个顶级目录：包含服务的实际配置的 **modules 目录**和包含每个环境的根配置的 **environments 目录**。

- modules 目录中为抽象出来的可复用模块。
- environments 目录用于隔离不同环境，用户可以在不同环境中使用不同的云厂商或配置不同的账号以实现多云管理（prod 目录中还通过 workspace 进行业务层面的隔离）。

以下是推荐的项目结构，详细内容请参见 Coding、Github 代码仓库。

```
.
├── README.md
├── environments
│   ├── dev
│   │   ├── main.tf
```

```
│   │   └── provider.tf
│   └── prod
│       ├── cicd
│       │   └── main.tf
│       ├── local.tf
│       ├── main.tf
│       ├── provider.tf
│       └── qta
│           └── main.tf
└── modules
    ├── network
    │   ├── main.tf
    │   ├── outputs.tf
    │   ├── provider.tf
    │   └── variables.tf
    ├── security_group
    │   ├── main.tf
    │   ├── outputs.tf
    │   ├── provider.tf
    │   └── variables.tf
    └── tke
        ├── main.tf
        ├── outputs.tf
        ├── provider.tf
        └── variables.tf
```

代码复用

若您希望代码复用，建议您使用 Terraform Module 。您可以选择使用自定义或使用供应商提供的 Module，例如腾讯云官方 Module。在该项目 Modules 目录中的已封装 Module 模板，通过对资源进行分类，将每一类资源用一个目录管理。然后在一个模板中管理目标资源，完成目标资源和依赖资源的串联，例如 TKE 模板中依赖 Network 资源模板。

构建安全的 Terraform 更新流程

确保基础架构的安全性取决于是否有一个稳定安全的应用 Terraform 更新的流程。

先规划再执行

始终先为 Terraform 执行生成计划。将计划保存到输出文件中。获得基础架构所有者的批准后，执行计划。即使开发者在本地对更改进行原型设计，也应该生成计划并查看应用添加、修改和销毁的资源。

实现自动化流水线

为了确保一致的执行上下文，通过自动化工具执行 Terraform。避免人为因素的影响。

避免导入现有资源

- 请尽可能避免导入现有资源（使用 terraform import），因为这样做可能会很难完全了解手动创建的资源的来源和配置。应通过 Terraform 创建新资源并删除旧资源。
- 如果删除旧资源会带来大量重复劳动，请使用 terraform import 命令并明确批准。将资源导入 Terraform 后，只能使用 Terraform 对其进行管理。

不要手动修改 Terraform 状态

若您在使用 Terraform 管理基础设施时又手动在控制台更新资源属性，则会导致 Terraform 执行计划时出现意外结果。如果您需要修改 Terraform 状态信息，请使用命令 `terraform state` 。

版本控制

与其他形式的代码类似，您可以将基础架构代码存储在版本控制系统中以保留历史记录并允许轻松回滚。

环境隔离

支持的隔离方式如下：

目录隔离

通过不同目录进行分组，在子目录中分别配置跟模块实现隔离（本文示例使用该方式）。
使用说明：environments 中的 dev 和 prod 目录使用不同的根模块隔离不同环境的基础设施配置信息。

branch 隔离

不同环境使用不同分支，在对应分支的根模块下初始化 terraform 即可。
使用说明：与目录隔离类似。不同环境使用不同的分支，直接在不同分支中使用根模块配置特定环境信息，通过合并不同分支的更改来部署修改。

workspace 隔离

相同根模块下通过配置不同 workspace 来分别创建资源。
使用说明：prod 目录中的 cicd 和 qta 使用 workspace 隔离不同业务的配置信息。

版权所有：腾讯云计算（北京）有限责任公司

持续集成与部署
Terraform 在 Github 中的应用

最近更新时间：2023-07-20 17:40:21

本文介绍如何将 Terraform 结合 Github action 实现自动化部署。

前置条件

1. 注册 Github 账号。
2. 注册腾讯云账号，并完成 实名认证。
3. 获取凭证，在 API密钥管理 页面中创建并复制 SecretId 和 SecretKey。

创建项目

在 GitHub 中新建代码仓库，目录结构如下：

```
.
├── README.md
├── environments
│   ├── dev
│   │   ├── main.tf
│   │   └── provider.tf
│   └── prod
│       ├── cicd
│       │   └── main.tf
│       ├── local.tf
│       ├── main.tf
│       ├── provider.tf
│       └── qta
│           └── main.tf
└── modules
    ├── network
    │   ├── main.tf
    │   ├── outputs.tf
    │   ├── provider.tf
    │   └── variables.tf
    ├── security_group
    │   ├── main.tf
    │   ├── outputs.tf
    │   ├── provider.tf
    │   └── variables.tf
    └── tke
```

```
        ├── main.tf
        ├── outputs.tf
        ├── provider.tf
        └── variables.tf
```

目录结构说明：

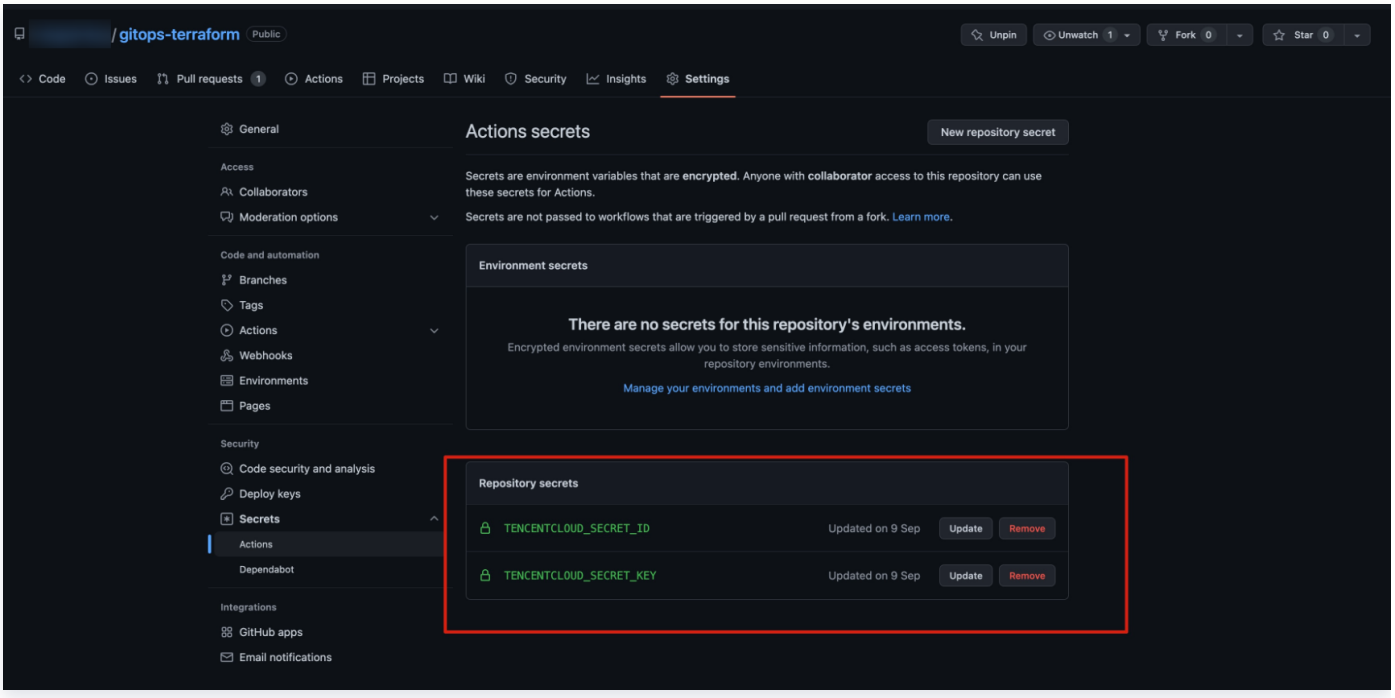
1. 项目结构主要分为 `environments` 和 `modules` 两个目录。
2. `environments` 为目录方式隔离环境 `dev` 和 `prod`，用来给不同环境设置各自的配置，每个环境目录都是独立的根模块。

> **说明：**
> - dev 中演示创建一个 vpc。
> - prod 中演示通过 workespace 进行业务隔离。在 cicd 目录中创建 vpc，在 qta 目录下创建容器集群。

3. `modules` 为封装的资源信息，用以复用。本目录中包含 vpc、安全组和容器服务 TKE 的 Module 演示。
4. 完整代码请参考 gitops-terraform 。

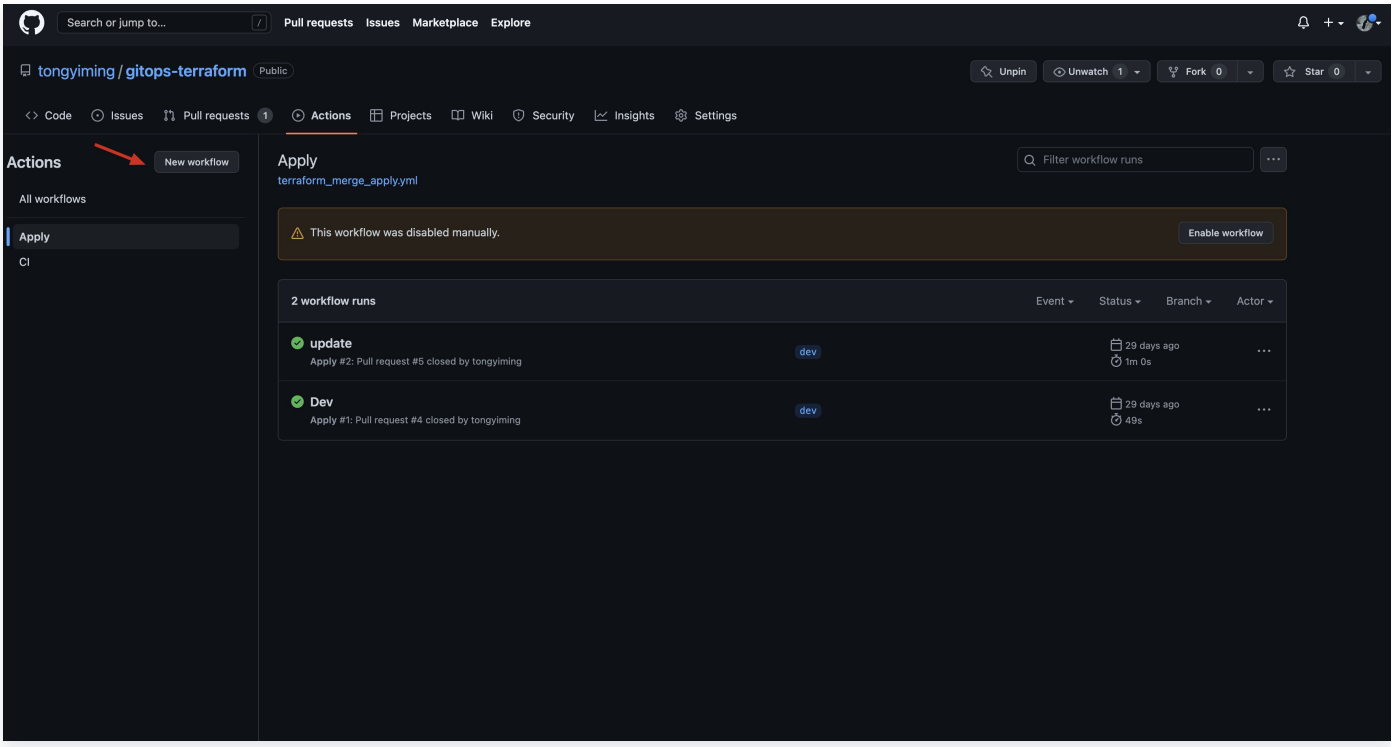
流水线配置

1. 为防止 AKSK 等安全信息泄流造成安全问题，您需要在 `https://github.com/${USER}/${PROJECT}/settings/secrets/actions` 设置环境变量。请替换为已复制的 SecretId 和 SecretKey。



2. 通过 GitHub Actions 配置流水线。

您可以在项目中的 actions 选项处单击 **New workflow**，也可以在 `.github/workflows/` 目录里通过添加 yml 文件创建，流水线配置详情可以参考 相关操作 。

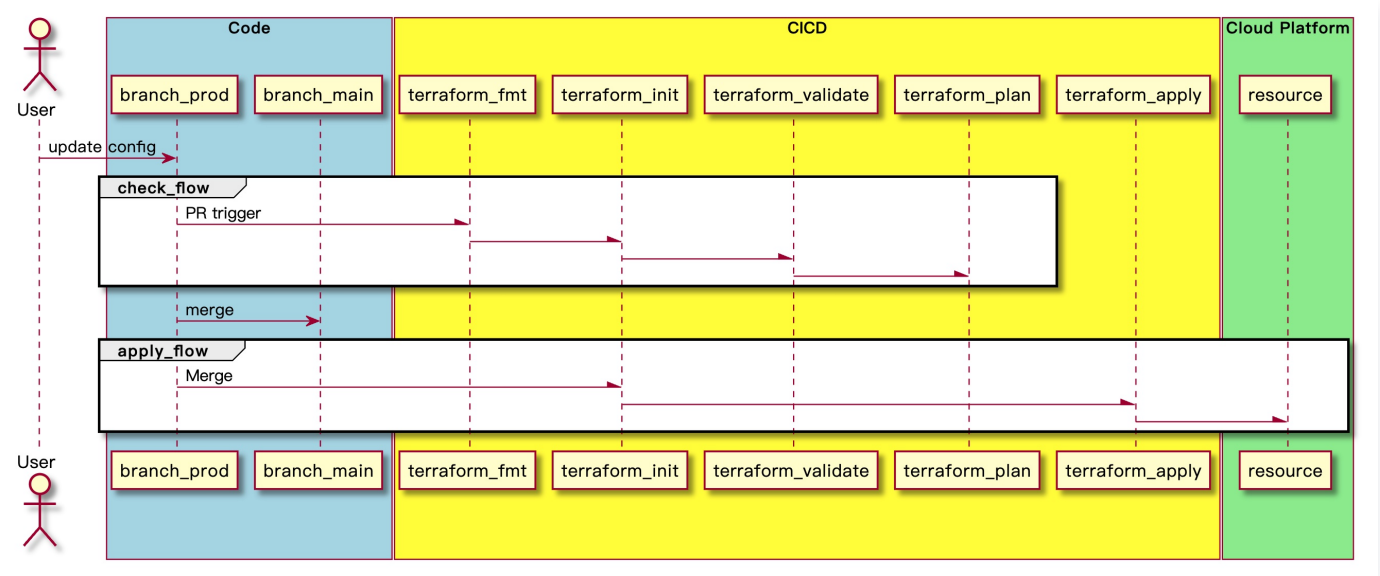


检查流水线

1. Terraform 根模块资源不能过多。同理，在执行检查的时候也应该尽可能的避免全部资源的读取，需要以细粒度的方式触发检查。
2. 本文档采用按分支区分触发的环境。例如，dev 中的配置需要更新时，只能在 dev 分支上进行更新。配置更新完成后提交 PR 将代码合入到 main（主分支）。目的是每次更新的检查不需要全量扫描 environments 下的所有子目录（环境），减少不必要的状态同步消耗。
3. 该流水线主要通过执行 `terraform fmt` 、 `terraform init` 、 `terraform validate` 、 `terraform plan` 来检查代码和展示构建计划，方便判断是否执行部署。

部署流水线

1. 如果检查流水线中的检查操作都成功且 `terraform plan` 的输出符合预期，那么就可以进行 merge 操作。
2. 在 merge 完成的时候触发部署（即 `terraform apply` ）的操作。示意图如下：



相关操作

在进入 environment 的指定环境目录后，会判断是否还有子目录，如果有则通过 workspace 隔离不同业务环境（例如 qta，ci），如果没有则等价于普通的根模块。

创建校验流水线

参考以下代码，下载 Terraform 和校验 Terraform 代码：

```
# This is a basic workflow to help you get started with Actions

name: CI

# Controls when the workflow will run
on:
  pull_request:


# A workflow run is made up of one or more jobs that can run sequentially
or in parallel
jobs:
  # This workflow contains a single job called "build"
  build:
    # The type of runner that the job will run on
    runs-on: ubuntu-latest
    env:
      TENCENTCLOUD_SECRET_KEY: ${{ secrets.TENCENTCLOUD_SECRET_KEY }}
      TENCENTCLOUD_SECRET_ID: ${{ secrets.TENCENTCLOUD_SECRET_ID }}
```

```
    # Steps represent a sequence of tasks that will be executed as part of
the job
    steps:
      - uses: actions/checkout@v3
      - uses: hashicorp/setup-terraform@v2
        with:
          terraform_wrapper: false

      - name: check env
        run: |
          if [ ! -d "environments/$GITHUB_HEAD_REF" ]; then
            echo
"************************SKIPPING***********************************"
            echo "Branch '$GITHUB_HEAD_REF' does not represent an oficial
environment."
            echo
"*******************************************************************"
            exit 1
          fi

      - name: terraform fmt
        id: fmt
        run: terraform fmt -recursive -check

      - name: terraform init
        id: init
        working-directory: environments/${{ github.head_ref    }}
        run: terraform init

      - name: terraform validate
        id: validate
        working-directory: environments/${{ github.head_ref    }}
        run: terraform validate

      - name: terraform plan
        id: plan
        if: github.event_name == 'pull_request'
        working-directory: environments/${{ github.head_ref    }}
        run: |
          plan_info=""
          dir_count=`ls -l | grep "^d" | wc -l`
          if [ $dir_count -gt 0 ]; then
            for dir in ./*/
            do
```

```
              env=${dir%*/}
              env=${env#*/}
              echo ""
              echo "========> Terraform Plan <========"
              echo "At environment: ${{ github.head_ref    }}"
              echo "At workspace: ${env}"
              echo "================================="
              terraform workspace select ${env} || terraform workspace new
${env}
              plan_info="$plan_info\n$(terraform plan -no-color)"
            done
          else
            plan_info="$(terraform plan -no-color)"
          fi
          plan_info="${plan_info//'%'/'%25'}"
          plan_info="${plan_info//$'\n'/'%0A'}"
          plan_info="${plan_info//$'\r'/'%0D'}"
          echo "::set-output name=plan_info::$plan_info"
        continue-on-error: true

      - uses: actions/github-script@v6
        if: github.event_name == 'pull_request'
        with:
          script: |
            const output = `#### Terraform Format and Style \`${{
steps.fmt.outcome }}\`
            #### Terraform Initialization \`${{ steps.init.outcome }}\`
            #### Terraform Validation \`${{ steps.validate.outcome }}\`
            #### Terraform Plan \`${{ steps.plan.outcome }}\`
            <details><summary>Show Plan</summary>
            \`\`\`\n
            ${{ steps.plan.outputs.plan_info }}
            \`\`\`
            </details>
            *Pushed by: @${{ github.actor }}, Action: \`${{
github.event_name }}\`*`;
            github.rest.issues.createComment({
              issue_number: context.issue.number,
              owner: context.repo.owner,
              repo: context.repo.repo,
              body: output
            })
```

创建部署流水线

```
name: Apply

on:
  pull_request:
    types:
      - closed
    branches:
      - main


jobs:
  build:
    if: github.event.pull_request.merged == true
    runs-on: ubuntu-latest
    env:
      TENCENTCLOUD_SECRET_KEY: ${{ secrets.TENCENTCLOUD_SECRET_KEY }}
      TENCENTCLOUD_SECRET_ID: ${{ secrets.TENCENTCLOUD_SECRET_ID }}

    steps:
      - uses: actions/checkout@v3
      - uses: hashicorp/setup-terraform@v2

      - name: terraform init
        id: init
        working-directory: environments/${{ github.head_ref    }}
        run: terraform init

      - name: terraform apply
        working-directory: environments/${{ github.head_ref }}
        run: |
          dir_count=`ls -l | grep "^d" | wc -l`
          if [ $dir_count -gt 0 ]; then
            for dir in ./*/
            do
              env=${dir%*/}
              env=${env#*/}
              echo ""
              echo "========> Terraform Apply <========"
              echo "At environment: ${{ github.head_ref    }}"
              echo "At workspace: ${env}"
              echo "================================="

              terraform workspace select ${env} || terraform workspace new
${env}
```

```
      terraform apply -auto-approve
    done
  else
    terraform apply -auto-approve
  fi
```